PATENT
5681-65900/P9386

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application No.: 10/600,906	§	Examiner:	Cao, Diem K.
Filed: June 20, 2003	§	Group/Art Unit:	2194
Inventors:	§	Atty. Dkt. No:	5681-65900
Jerome M. Verbeke	§		
Neelakanth M. Nadgir	§		
	§		
Title: DYNAMIC LOADING OF REMOTE CLASSES	§		

APPEAL BRIEF

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir/Madam:

Further to the Notice of Appeal mailed September 25, 2008, Appellants present this Appeal Brief. Appellants respectfully request that the Board of Patent Appeals and Interferences consider this appeal.

I. REAL PARTY IN INTEREST

The subject application is owned by Sun Microsystems, Inc., a corporation organized and existing under and by virtue of the laws of the State of Delaware, and now having its principal place of business at 4150 Network Circle, Santa Clara, CA 95054.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF CLAIMS

Claims 1-9, 11-27, 29-43, 45-60 and 62-68 are currently pending in this case. Claims 10, 28, 44, and 61 were previously cancelled. All of the pending claims stand finally rejected and are the subject of this appeal. A copy of the claims, as incorporating entered amendments and as on appeal, is included in the Claims Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments have been filed subsequent to the rejection in the Final Office Action of June 25, 2008. The Claims Appendix hereto reflects the current state of the claims.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Independent claim 1 recites a system which includes a processor and a memory medium comprising program instructions. *See, e.g., at least page 10, line 21-page 11, line 14; Figure 1 (e.g., processor 102 and memory 104).* The program instructions are executable to implement a virtual machine. *See, e.g., at least page 11, lines 9-14; page 12, lines 3-18; page 13, lines 9-25; Figure 1.* The program instructions are further executable to implement a default class loader for the virtual machine. The default class loader is configured to load classes for code executable within the virtual machine on the system from one or more local locations indicated by a class path of the default class loader. *See, e.g., at least page 4, lines 4-11; page 9, lines 4-8; page 9, lines 20-24; page 11, lines 9-24; page 12, lines 3-9; page 12, lines 20-24; page 15, lines 12-15; Figures 1-2D, 4 (114).* The default class loader is further configured to determine that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path. The default class loader is also configured to generate an indication that the class is not loaded. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The program instructions are also executable to implement a remote class loader mechanism. The remote class loader is configured to detect the indication that the class is not loaded. *See, e.g., at least page 9, lines 3-9; page 11, lines 26-29; Figures 1-2D, 4 (110); Figure 3 (200).* The remote class loader is further configured to obtain the class from a remote system via a network. The remote class loader is also configured to store the class in a location indicated by the class path of the default class loader on the system. *See, e.g., at least page 9, lines 10-13; page 9, line 24-page 10, line 19; page 11, lines 26-29; page 12, lines 13-18; Figures 1-2D, 4 (110); Figure 3 (202, 204).* The remote class loader mechanism is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader, and the default class loader is independent from the remote class loader mechanism. *See, e.g., at least page 9, lines 3-13; page 11, line 25 – page 12, line 19; Figures 1, 2A-2D, and 4 (110, 114).* The default class loader is configured to load the class from the location indicated by the class path and the default class loader being

configured to load the class from the location avoids class conflicts. *See, e.g., at least page 3, lines 5-11; page 9, lines 3-6; page 9, lines 11-12; page 9, lines 17-27.*

Independent claim 18 recites a distributed computing system which includes a master node configured to provide computer-executable code fragments of an application to a plurality of worker nodes on a network, wherein the code fragments are configured to run tasks in parallel on two or more of the plurality of worker nodes to perform a job. The distributed computing system further includes a worker node configured to receive a code fragment from the master peer node. *See, e.g., at least page 17, lines 4-16; page 18, lines 19-20; Figure 4 (130, 132, 134, 136).*

The worker node includes a virtual machine and a default class loader for the virtual machine. The default class loader is configured to load classes for the code fragment executable within the virtual machine from one or more local locations indicated by a class path of the default class loader. For a virtual machine, s*ee, e.g., at least page 11, lines 9-14; page 12, lines 3-18; page 13, lines 9-25; Figure 1.* Also *see, e.g., at least page 4, lines 4-11; page 9, lines 4-8; page 9, lines 20-24; page 11, lines 9-24; page 12, lines 3-9; page 12, lines 20-24; page 15, lines 12-15; Figures 1-2D, 4 (114).* The default class loader is further configured to determine that a class needed to execute the code fragment is not stored in the one or more locations indicated by the class path. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The worker node further comprises a remote class loader configured to detect that the class is not loaded. *See, e.g., at least page 9, lines 3-9; page 11, lines 26-29; Figures 1-2D, 4 (110); Figure 3 (200).* The remote class loader is further configured to obtain the class from a remote node via the network and store the class in a location indicated by the class path of the default class loader on the worker node. *See, e.g., at least page 9, lines 10-13; page 9, line 24-page 10, line 19; page 11, lines 26-29; page 12, lines 13-18; Figures 1-2D, 4 (110); Figure 3 (202, 204).* The remote class loader is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader, and the default class loader is independent from the remote class loader. *See, e.g., at least page 9, lines 3-13; page 11, line 25 – page 12,*

line 19; Figures 1, 2A-2D, and 4 (110, 114). The default class loader is further configured to load the class from the location indicated by the class path, and the default class loader being configured to load the class from the location avoids class conflicts. *See, e.g., at least page 3, lines 5-11; page 9, lines 3-6; page 9, lines 11-12; page 9, lines 17-27.*

Independent claim 34 recites a system which includes a default class loader means for a virtual machine. For a virtual machine, s*ee, e.g., at least page 11, lines 9-14; page 12, lines 3-18; page 13, lines 9-25; Figure 1.* The default class loader means is configured to load classes for code executable within the virtual machine on the system from one or more local locations indicated by a class path of the default class loader means. *See, e.g., at least page 4, lines 4-11; page 9, lines 4-8; page 9, lines 20-24; page 11, lines 9-24; page 12, lines 3-9; page 12, lines 20-24; page 15, lines 12-15; Figures 1-2D, 4 (114).* The system further includes means for determining that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The system further includes means for obtaining the class from a remote system via a network and means for storing the class in a location on the system indicated by the class path of the default class loader means. *See, e.g., at least page 9, lines 3-13; page 9, line 24-page 10, line 19; page 11, lines 26-29; page 12, lines 13-18; Figures 1-2D, 4 (110); Figure 3 (202, 204).* Said means for determining, said means for obtaining, and said means for storing are configured to operate separate from and transparent to the default class loader, and the default class loader means is independent from said means for determining, said means for obtaining, and said means for storing. *See, e.g., at least page 9, lines 3-13; page 11, line 25 – page 12, line 19; Figures 1, 2A-2D, and 4 (110, 114).* The default class loader means is configured to load the class from the location indicated by the class path, and the default class loader means being configured to load the class from the location avoids class conflicts. *See, e.g., at least page 3, lines 5-11; page 9, lines 3-6; page 9, lines 11-12; page 9, lines 17-27.*

Independent claim 35 recites a method comprising loading classes for code executing within a virtual machine on a system from one or more local locations indicated by a class path of a default class loader for the virtual machine. S*ee, e.g., at least page 11, lines 9-14; page 12, lines 3-18; page 13, lines 9-25; Figure 1.* Also, *see, e.g., at least page 4, lines 4-11; page 9, lines 4-8; page 9, lines 20-24; page 11, lines 9-24; page 12, lines 3-9; page 12, lines 20-24; page 15, lines 12-15; Figures 1-2D, 4 (114).* The method further comprises determining that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The method further comprises generating an indication that the class is not loaded. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The method further comprises detecting the indication that the class is not loaded. *See, e.g., at least page 9, lines 3-9; page 11, lines 26-29; Figures 1-2D, 4 (110); Figure 3 (200).* The method further comprises obtaining the class from a remote system via a network in response to said detecting, and storing the class in a location indicated by the class path of the default class loader on the system. *See, e.g., at least page 9, lines 10-13; page 9, line 24-page 10, line 19; page 11, lines 26-29; page 12, lines 13-18; Figures 1-2D, 4 (110); Figure 3 (202, 204).* Said detecting, said obtaining, and said storing are performed separate from and transparent to the default class loader, and the default class loader is independent from said detecting, said obtaining, and said storing. *See, e.g., at least page 9, lines 3-13; page 11, line 25 – page 12, line 19; Figures 1, 2A-2D, and 4 (110, 114).* The method further comprises the default class loader loading the class from the location indicated by the class path, wherein the default class loading the class from the location avoids class conflicts. *See, e.g., at least page 3, lines 5-11; page 9, lines 3-6; page 9, lines 11-12; page 9, lines 17-27.*

Independent claim 52 recites a computer-accessible storage medium comprising program instructions computer-executable to implement the method as recited in claim 35. The program instructions are computer-executable to implement loading classes for code executing within a virtual machine on a system from one or more local locations indicated by a class path of a default class loader for the virtual machine. S*ee, e.g., at*

least page 11, lines 9-14; page 12, lines 3-18; page 13, lines 9-25; Figure 1. Also, *see, e.g., at least page 4, lines 4-11; page 9, lines 4-8; page 9, lines 20-24; page 11, lines 9-24; page 12, lines 3-9; page 12, lines 20-24; page 15, lines 12-15; Figures 1-2D, 4 (114).* The program instructions are further computer-executable to implement determining that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The program instructions are further computer-executable to implement generating an indication that the class is not loaded. *See, e.g., at least page 11, lines 19-26; page 12, lines 8-10; page 15, lines 12-15; Figures 1-2D, 4 (114).* The program instructions are further computer-executable to implement detecting the indication that the class is not loaded. *See, e.g., at least page 9, lines 3-9; page 11, lines 26-29; Figures 1-2D, 4 (110); Figure 3 (200).* The program instructions are further computer-executable to implement obtaining the class from a remote system via a network in response to said detecting, and storing the class in a location indicated by the class path of the default class loader on the system. *See, e.g., at least page 9, lines 10-13; page 9, line 24-page 10, line 19; page 11, lines 26-29; page 12, lines 13-18; Figures 1-2D, 4 (110); Figure 3 (202, 204).* Said detecting, said obtaining, and said storing are performed separate from and transparent to the default class loader, and the default class loader is independent from said detecting, said obtaining, and said storing. *See, e.g., at least page 9, lines 3-13; page 11, line 25 – page 12, line 19; Figures 1, 2A-2D, and 4 (110, 114).* The program instructions are further computer-executable to implement the default class loader loading the class from the location indicated by the class path, wherein the default class loading the class from the location avoids class conflicts. *See, e.g., at least page 3, lines 5-11; page 9, lines 3-6; page 9, lines 11-12; page 9, lines 17-27.*

The summary above describes various examples and embodiments of the claimed subject matter; however, the claims are not necessarily limited to any of these examples and embodiments. The claims should be interpreted based on the wording of the respective claims.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Claims 1-8, 11-13, 16, 34-42, 45-47, 50, 52-59, 62-64 and 67 stand finally rejected under 35 U.S.C. § 103(a) as being unpatentable over Monday et al. (U.S. Patent 6,263,377) (hereinafter "Monday") in view of Venners ("Inside the Java Virtual Machine").

2. Claims 9, 14, 15, 17-27, 29-33, 43, 48, 49, 51, 60, 65, 66 and 68 stand finally rejected as being unpatentable over Monday in view of Venners and further in view of Babaoglu et al. ("Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems") (hereinafter "Babaoglu").

VII. ARGUMENT

First Ground of Rejection

Claims 1-8, 11-13, 16, 34-42, 45-47, 50, 52-59, 62-64 and 67 stand finally rejected under 35 U.S.C. § 103(a) as being unpatentable over Monday et al. (U.S. Patent 6,263,377) (hereinafter "Monday") in view of Venners ("Inside the Java Virtual Machine"). Appellants respectfully traverse this rejection for at least the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 1- 3, 5, 11-13, 34- 37, 39, 45-47, 52-54, 56, and 62-64

1. The cited references fail to disclose, alone or in combination, a remote class loader mechanism configured to: detect the indication that the class is not loaded; obtain the class from a remote system via a network; and store the class in a location indicated by the class path of the default class loader on the system; wherein the remote class loader mechanism is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader.

In claim 1 of the present application, *detecting the indication that the class is not loaded,* said *obtaining the class from a remote system via a network,* and said *storing the class in a location on the system indicated by the class path of the default class loader mechanism* are all performed separately from and transparently to the default class loader for the virtual machine. The default class loader attempts to load a class from a class path of the default class loader mechanism. If the class is not located, the recited remote class loader detects the indication that the class needed to execute the code on the system is not stored in the one or more locations indicated by the class path, obtains the class from a remote system via a network, and stores the class in a location on the system indicated by the class path of the default class loader mechanism. These operations are separate from and transparent to the default class loader. The default class loader may then load the

class from the location on the system indicated by the class path. There is no indication of this transparency in the cited references.

Page 1, line 10-page 3, line 5 of the Background section of the instant application generally describes the dynamic loading of classes using class loaders in the prior art. In page 1 line 28 – page 2 line 12, Appellants describe the concept and general operations of custom class loaders in virtual machines. The first part of this section states:

> Virtual machines such as JVM may provide a facility by which a user can introduce a custom class loader. For example, in JVM, a hook is provided to the loading mechanism through the custom class loaders. Programmatically speaking, [custom] class loaders are ordinary objects that may be defined in code (e.g. Java™ code). In Java™, [custom] class loaders are instances of subclasses of abstract class Classloader.

A conventional mechanism for dynamically loading classes not available on the class path of the default class loader for a virtual machine is via custom class loaders, as is described on page 3, lines 4-5 of the present application:

> In Java, to use a class, the class has to be in the class path of the default class loader, or alternatively a custom class loader may be provided.

Claim 1 of the instant application recites a remote class loader mechanism to remotely load classes needed to run, for example, an application in a distributed computing environment through the default class loader of the virtual machine (i.e., not through a custom class loader to remotely load the classes) (see, for example, page 3, lines 9-11 and page 4, lines 3-11 of the present application). In contrast, what the Monday reference describes in col. 3, lines 38-56 and elsewhere is the use of "remoteclassloader", which clearly is a custom class loader as disclosed in the background section of the present application. In the cited section, Monday states "If x.class is NOT located, then a **subclass of** the classloader, a remoteclassloader…" In the background section of the present application, as noted above, Appellants explain that custom class loaders are instances of **subclasses** of abstract class Classloader. When Monday refers to remoteclassloader, Monday is referring to a custom class loader. Monday's remoteclassloader clearly operates as a custom class loader in the conventional method as described in the background section of the present application to load classes

not found on the class path of Monday's classloader.

Thus, what Monday discloses is clearly and distinctly different than what is recited in claim 1 of the present application. Monday's system is actually described as operating according to the conventional method for loading classes that relies on custom class loaders. Monday's remoteclassloader is simply a custom class loader as described in Appellants' background section. Appellants' specification discloses a method for loading classes that intentionally does not rely on the conventional method in virtual machines using custom class loaders to avoid problems created thereby; instead, Appellants' disclosed method uses a remote class loader mechanism that transparently determines or detects that a class has not been loaded from a location on the class path of the default class loader, locates the class, obtains the class, and stores the class in the location indicated by the class path of the default class loader without using custom class loaders to load the class. The default class loader itself may then load the class from the location indicated by the class path of the default class loader. This is the system recited in claim 1.

In summary, Monday describes a method that relies on a custom class loader to load a class, while claim 1 recites a system for implementing a method for remotely loading classes that does **not** rely on a custom class loader to load the class. Further, Monday does not teach a method in which detecting an indication that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path, obtaining the class from a remote system via a network, and storing the class in a location on the system indicated by the class path of the default class loader, are performed separately from and transparently to the default class loader for the virtual machine. In col. 3, lines 38-56 and elsewhere, Monday states "If x.class is NOT located, then a **subclass of** the classloader, a remoteclassloader…". **By definition, since Monday's remoteclassloader is a subclass of Monday's classloader, it cannot and does not operate separately from and transparently to the classloader.** Clearly, Monday does not anticipate claim 1 of the instant application.

The Examiner has asserted, with respect to this limitation, "inherent from the remote class loader check the remoteclasspath, obtain the class and store it in the directory without consulting from the class loader; col. 2, lines 44-56". **The Examiner's assertion is entirely unsupported by the actual teachings of the reference**. Instead, Applicant submits that the remoteclassloader is likely invoked by the default classloader in the event that the class is not found in the classpath ("A CLASS is requested from the remoteclassloader", col. 4, lines 22-23. Appellants respectfully submit that it is clearly not inherent that Monday's remoteclassloader operates both separately from and transparently with respect to the default classloader. According to M.P.E.P. 2131.01 III, in regard to a theory of inherency "evidence must make clear that the missing descriptive matter is **necessarily** present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill." The Examiner has not provided anything but the Examiner's own unsupported speculation that the functionality relied upon by the Examiner is inherent in Monday. The rejection is not supported by the actual evidence of record.

In the Action dated June 25, 2008, the Examiner essentially reiterates the same points as before, and fails to substantially address numerous ones of the arguments previously presented in this case. For example, in response to the above arguments, the Examiner merely makes the following immaterial assertion:

> …it is noted that the features upon which applicant relies (i.e., the remote class loader is not a custom class loader) are not recited in the rejected claim(s)…In addition, the remote class loader in the system of Monday performs its functions without communication with the default class loader, clearly, the remote class loader operate separately from and transparently to the default class loader. [Sic]

Appellants respectfully submit that comments regarding a custom class loader were provided to distinguish Appellants' current claimed limitations from the prior art (those described in Monday, Venners, and Appellants' own background section). As already noted, custom class loaders are instances of subclasses of the Classloader, and by definition, and as well know by any one of ordinary skill in the art of class loaders, cannot operate separately from and transparently to the default classloader and cannot be

independent from the default classloader (recited in the claims). Furthermore, Appellants have already addressed the Examiner's unsupported conclusory statement that the remote class loader of Monday "clearly" operates separately from and transparently to the default class loader. Additionally, the Examiner does not address arguments regarding the Examiner's assertions of inherency, which are clearly unsupported by the prior art and violate the guidelines set forth in the MPEP. Appellants respectfully submit that the Examiner fails to address these arguments and that Appellants' statements regarding custom class loaders were provided as indications of the prior art, over which Appellants' current claim limitations clearly distinguish. Thus, the Examiner has not substantially addressed Appellants' arguments or provided *a prima* facie case of obviousness for the present claims.

In the Advisory Action dated September 12, 2008, the Examiner asserts, in regard to the above arguments (point 1), "Applicant failed to provide any reasons why the cited passages do not teach the limitations 'a remote class loader mechanism configured to: detect the indication that the class is not loaded; obtain the class from a remote system via a network; and store the class in a location indicated by the class path of the default class loader on the system', therefore, the arguments regarding the above limitations are not persuasive." **Appellants have <u>clearly</u> provided reasons, in previous Actions and in this Appeal, as to why the cited references do not teach these limitations.** The Examiner actually contradicts himself in stating both "the arguments regarding the above limitations" and "Applicant failed to provide any reasons why the cited passages do not teach the limitations". Appellants' arguments in response to the Examiner's assertions regarding the limitations are obviously Appellants' reasons in regards to why the cited art does not teach the limitations.

In the Advisory Action dated September 12, 2008, in further regard to the above arguments (point 1), the Examiner further asserts "any one of ordinary skill in the art would have known that even though the custom class loaders are instances of subclasses of the Classloader, the default class loader and the custom class loader are separate entities, and no definition shows that the remote [custom] class loader operates must

depend [sic] on the default class loader." The Examiner's interpretation "remote class loader operates must depend [sic] on the default class loader" is not the actual argument of Appellants. Monday clearly teaches that Monday's remoteclassloader is "a **subclass of** the classloader" in Monday's system. Monday's remoteclassloader is thus clearly consistent with the definition of a custom class loader. A custom classloader definition *extends* the classloader definition (hence, "subclass"). A custom classloader must be (and is, by definition) aware of its parent class. In Monday's system, when remoteclassloader is instanced, by definition, its parent class, classloader, is instanced, since remoteclassloader is a subclass of classloader.

Moreover, one of ordinary skill in the art would understand that the operations of custom classloaders and the default classloader in, for example, a Java environment/JVM, are necessarily performed in concert. Appellants refer the Examiner to any good tutorial on class loaders in, for example, the Java programming environment, to understand the operations of class loaders including custom class loaders. **The Venners reference, cited by the Examiner, is one example**. The Examiner requested from Appellants "all references/information that teach…". It is not reasonable to insist on "**all** references/information that teach…". Further, a thorough reading of the Monday reference clearly shows that Monday's classloader/remoteclassloader operate and interact in accordance with the common descriptions of default classloader/custom classloaders.

The Examiner goes on to assert "When the default class loader fail to find a class, it will throw an exception, and that is the end of the execution of the default class loader when attempt to load the class, then the remote class loader catch the exception and executes its codes to try to find the requested class. Clearly, the remote class loader is executed transparently and independent of the default class loader." First, the Examiner clearly does not understand the operations of default and custom classloaders, for example in the Java programming environment. The default and custom classloaders in such an environment do not operate in the way described by the Examiner. A custom classloader definition *extends* the classloader definition (hence, "subclass"). A custom classloader must be (and is, by definition) aware of its parent class, and interacts directly

with its parent class. In Monday's system, when remoteclassloader is instanced, by definition, its parent class, classloader, is instanced, since remoteclassloader is a subclass of classloader. The Examiner's assertion "the remote class loader is executed transparently and independent of the default class loader" is clearly without basis.

Moreover, the Monday reference clearly does not teach what the Examiner asserts (""When the default class loader fail to find a class, it will throw an exception, and that is the end of the execution of the default class loader when attempt to load the class, then the remote class loader catch the exception and executes its codes to try to find the requested class."). **In addition, the Venners reference, relied upon in part by the Examiner, clearly does not teach "the remote class loader is executed transparently and independent of the default class loader."** In fact, Venners teaches the opposite: a user-defined class loader must execute in concert with (or within view, as opposed to transparently) the default class loader (e.g., bootstrap or system class loader), and user-defined class loaders are not independent of the default class loader (e.g., bootstrap or system class loader). Venners teaches in Chapter 5, pages 11-12, under the heading "User-Defined Class Loaders", that "Although user-defined class loaders themselves are part of the Java application, four of the methods in ClassLoader are gateways into the Java virtual machine...**Any Java virtual machine implementation must take care to connect these methods of class ClassLoader to the internal class loader subsystem**." (Emphasis added). Venners goes on to teach "When a user-defined class loader invokes [findSystemClass()] in version 1.0 and 1.1, **it is requesting that the virtual machine attempt to load the named type via its bootstrap class loader**...in version 1.2., the findSystemClass() method **attempts to load the requested type from the system class loader**. Every Java virtual machine implementation must make sure the findSystemClass() method can invoke the [bootstrap or system] class loader in this way."

In the Advisory Action dated September 12, 2008, the Examiner asserts, in regard to point 2, "examiner fails to find in the reference that '**the remote class loader is called by the class loader** in the event that the class is not found in the class path' as asserted by Applicant. Therefore, the arguments are not persuasive." Appellants note

that the actual argument was stated as "the remoteclassloader is **likely** invoked by the default classloader in the event that the class is not found in the classpath." This argument is part of Appellants' arguments in response to Examiner's assertion that it is "inherent from the remote class loader check the remoteclasspath, obtain the class and store it in the directory without consulting from the class loader". **The Examiner's assertion is entirely unsupported by the actual teachings of the reference**. Moreover, Monday teaches "If x.class is NOT located, then a **subclass of** the classloader, a remoteclassloader…" (col. 3, lines 43-45) and "A CLASS is requested from the remoteclassloader" (col. 4, lines 22-23). It is clearly not inherent that Monday's remoteclassloader operates both separately from and transparently with respect to the default classloader. The Examiner has not provided anything but the Examiner's own unsupported speculation that the functionality relied upon by the Examiner is inherent in Monday.

In the Advisory Action dated September 12, 2008, the Examiner asserts, in regard to point 3, "examiner fails to find in the reference that teach 'the default class loader notifies a custom class loader' when the default class loader cannot locate a class in a location in its class path." Monday clearly teaches "If x.class is NOT located, then a **subclass of** the classloader, a remoteclassloader…" (col. 3, lines 43-45) and "A CLASS is **requested from** the remoteclassloader" (col. 4, lines 22-23), which is consistent with Appellants' assertion.

In the Advisory Action dated September 12, 2008, the Examiner further asserts, in regard to point 3, "the rejection clearly show 'the default class loader loads the class from the location indicated by the class path, wherein the class is stored therein by a remote class loader'," and goes on to assert "Applicant failed to give any reason why the cited passages do not teach the above limitation. Therefore, the arguments are not persuasive." Contrary to the Examiner's assertion, Appellants have provided reasons as to why the cited passages do not teach this limitation. Appellants refer the Examiner to, for example, Appellants' arguments under claim 1, subheading 2.

Furthermore, Appellants' note that it is the Examiner who shoulders the burden to establish a *prima facie* rejection. For the reasons given above, the Examiner has clearly failed to establish a *prima facie* case of obviousness.

2. The cited references fail to disclose, alone or in combination, wherein the default class loader is independent from the remote class loader mechanism.

As already indicated above, the remoteclassloader (which the Examiner currently relies on as the remote class loader) of Monday is a subclass of classloader. Thus, **by definition,** Monday's remoteclassloader is not independent from the default class loader. By definition, a subclass is not independent from its class.

In regards to Venners, page 1, line 10-page 3, line 5 of the Background section of the instant application generally describes the dynamic loading of classes using class loaders in the prior art. In particular, page 3, lines 4-5 describes that: "In Java, to use a class, the class has to be in the class path of the default class loader, or alternatively a **custom** class loader may be provided." The Venners reference is an overview of the Java Virtual Machine (JVM). Chapters 1-4 "give a broad overview of Java's architecture". A careful review of the Venners reference, particularly an overview given in Chapter 3, page 2, shows that Venners' description of Java's method of loading classes using class loaders is consistent with what is described in the Background section of the instant application. For example, Venners, in Chapter 3, page 2, gives the following description of how Java's class loading mechanism works:

> Imagine that during the course of running the Java application, a request is made of your [custom] class loader to load a class named Volcano. Your [custom] class loader would first ask its parent, the class path class loader, to find and load the class. The class path class loader, in turn, would make the same request of its parent, the installed extensions class loader. This class loader, would also first delegate the request to its parent, the bootstrap class loader. Assuming that class Volcano is not a part of the Java API, an installed extension, or on the class path, all of these class loaders would return without supplying a loaded class named Volcano. When the class path class loader responds that neither it nor any of its parents can load the class [i.e., the requested class is not on a class path], your [custom] class loader could then attempt to load the Volcano class in

> its custom manner, by downloading it across the network. Assuming your [custom] class loader was able to download class Volcano, that Volcano class could then play a role in the application's future course of execution.

In contrast to the above description from Venners, claim 1 of the instant application recites that a *default class loader for a virtual machine* determines that a class needed to execute code on the system is not stored in one or more local locations indicated by the class path (and thus fails to load the class). An indication is generated that the class is not loaded. A remote class loader mechanism then, transparently to the default class loader of the virtual machine, detects the indication, obtains the class from a remote system via a network, and stores the class in a location indicated by the class path. The default class loader (**not** a custom class loader, as in the Venners reference and the Monday reference, and as the Java class loading mechanism described by Venners and described in the Background section of the instant application operates) then loads the class from the location indicated by the class path.

To summarize, Venners discloses that, if the "default class loader" (**class path class loader**) cannot locate a class in a location on its class path, the "default class loader" notifies a "custom class loader" associated with the class, which then attempts to load the class, possibly from a remote location. Similarly, the Monday reference describes, in col. 3, lines 38-56, a remoteclassloader that clearly operates as a conventional custom class loader as is disclosed in Venners and in the background section of the present application. In contrast, claim 1 of the instant application recites that, if the "default class loader" cannot locate a class in a location on its class path, a "remote class loader mechanism", transparently to the default class loader, determines that the class was not located on the class path of the default class loader, locates the class on a remote system, and stores the class in a location indicated by the default class loader's class path. The default class loader then loads the class from the location. Thus, what claim 1 of the instant application recites is clearly distinct from the Java class loading mechanism as described by the Venners reference and from Monday's disclosed system.

In the Action dated June 25, 2008, in response to these arguments, the Examiner merely asserts that the present claims do not recite that the remote class loader is not a custom class loader and provides a conclusory statement regarding the transparent and separate operation of the remote class loader. **The Examiner apparently has failed to understand Appellants' argument.** The Examiner ignores the substance of Appellants' arguments regarding the limitation "wherein the default class loader is independent from the remote class loader mechanism" and does not provide any actual support for the conclusion that the remote class loader of Monday acts transparently to and separate from the default class loader. Appellants assert that the provision of unsupported and conclusory statements does not provide a *prima facie* case of obviousness.

3. The cited references fail to disclose, alone or in combination, wherein the default class loader being configured to load the class from the location avoids class conflicts.

Monday nowhere discloses this feature. There is no mention in either Monday or Venners as to avoiding class conflicts when loading from a particular location. Furthermore, as already indicated above, Venners teaches the use of custom class loaders for loading remote classes **which Appellants are specifically avoiding in claim 1**. As indicated in the background of the present application, using custom class loaders can lead to class conflicts. Accordingly, the cited art does not teach this feature of claim 1 and the Examiner has failed to establish a *prima facie* case of obviousness.

4. The Examiner has failed to provide a proper reason for combining the references.

The Examiner's assertion, **"**because Venners teaches the details how a Java application load and utilize class that are needed at runtime", is if anything simply a "motive" for understanding **how the Monday system already works**, since the Monday system **already employs** the conventional class loading method described in Venners. Furthermore, the present application describes in the background section the conventional

method for dynamically loading classes in virtual machines such as JVMs as is described in Venners, and clearly discloses a system and method that is **distinctly different** than the conventional method described in Venners. Claim 1 of the instant application **clearly** recites elements that enable the distinctly different method described in Appellants' specification. Moreover, the Examiner's stated reason is merely conclusory. Thus, the Examiner has failed to establish a *prima facie* case of obviousness.

Claims 4, 38, and 55

Claims 4, 38, and 55 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 4, 38, and 55 for the following reasons.

The Examiner has failed to establish that the cited references teach wherein the location is a user-specified directory for storing remote classes.

The Examiner simply asserts "Venners teaches the location is a user-specified directory for storing remote classes", and cites "user-defined directory path", chapter 5, page 11. However, **the citation clearly does not teach the limitation as recited in the claim.** The references simply do not describe that the location is a user-specified directory for storing remote classes.

Claims 6, 40, and 57

Claims 6, 40, and 57 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 6, 40, and 57 for the following reasons.

The Examiner has failed to establish that the cited references teach wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to send a message requesting the class to one or more remote systems, wherein the message comprises information about the class for identifying a class file on the remote system that comprises the requested class.

The Examiner simply asserts that Monday teaches the above limitations, and cites "a remoteclassloader checks…if the class is found", col. 3, lines 45-54. **However, neither this citation, nor elsewhere in Monday, teaches the claim limitations as recited in claim 6.** Instead, Monday specifically teaches a distinctly different method, for example at col. 3, lines 43-53, in which "ftp locations and./or machine names and directory paths" are maintained on the system in a REMOTECLASSPATH, and "distributed application manager 132 checks each server 122 in sequence for the particular selected x.class file". There would be no need in such a system to "send a message requesting the class to one or more remote systems, wherein the message comprises information about the class for identifying a class file on the remote system that comprises the requested class."

Claims 7, 41, and 58

Claims 7, 41, and 58 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 7, 41, and 58 for the following reasons.

The Examiner has failed to establish that the cited references teach wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to: send a message requesting the class to the remote system; and receive the class from the remote system in one or more messages in response to the message..

The Examiner simply asserts that Monday teaches the above limitations, and cites "a remoteclassloader checks…if the class is found", col. 3, lines 45-54. **However, neither this citation, nor elsewhere in Monday, teaches the claim limitations as recited in claim 7.** Instead, Monday specifically teaches a distinctly different method, for example at col. 3, lines 43-53, in which "ftp locations and./or machine names and directory paths" are maintained on the system in a REMOTECLASSPATH, and "distributed application manager 132 checks each server 122 in sequence for the particular selected x.class file".

Claims 8, 42 and 59

Claims 8, 42 and 59 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 8, 42 and 59 for the following reasons.

The Examiner has failed to establish that the cited references teach wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to: broadcast a message requesting the class to one or more remote systems including the remote system on the network; and receive the class from the remote system in one or more messages in response to the broadcast message.

The Examiner simply asserts that Monday teaches the above limitations, and cites "a remoteclassloader checks…if the class is found", col. 3, lines 45-54. **However, neither this citation, nor elsewhere in Monday, teaches the claim limitations as recited in claim 8.** Instead, Monday specifically teaches a distinctly different method, for example at col. 3, lines 43-53, in which "ftp locations and./or machine names and directory paths" are maintained on the system in a REMOTECLASSPATH, and "distributed application manager 132 checks each server 122 in sequence for the

particular selected x.class file". There would be no need in such a system to "broadcast a message requesting the class to one or more remote systems."

Claims 16, 50 and 67

Claims 16, 50 and 67 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 16, 50 and 67 for the following reasons.

The cited references do not disclose wherein the code is a code fragment of an application configured for execution on the system, and wherein the remote system is a node in a distributed computing framework that comprises the application and is configured to provide computer-executable code fragments of the application to two or more other systems to run the code fragments in parallel to execute the application.

The Examiner simply asserts that Monday teaches the above limitations, and cites "abstract." Neither Monday, nor the cited references in combination, teach the limitations as recited in the claims. Monday is not directed at a distributed computing framework as recited in the claims, nor does Monday teach anything like providing computer-executable code fragments of an application to two or more other systems to run the code fragments in parallel to execute the application.

Second Ground of Rejection

Claims 9, 14, 15, 17-27, 29-33, 43, 48, 49, 51, 60, 65, 66 and 68 stand finally rejected as being unpatentable over Monday in view of Venners and further in view of Babaoglu et al. ("Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems") (hereinafter "Babaoglu"). Appellants respectfully traverse this rejection for at

least the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 9, 43 and 60

Claims 9, 43 and 60 depend from claims 8, 42, and 59, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claims 8, 42 and 59. In addition, Appellants traverse the rejection of claims 9, 43 and 60 for the following reasons.

The Examiner has failed to establish that the cited references, alone or in combination, teach wherein the one or more remote systems and the system are member peers of a peer group in a peer-to-peer network environment.

The Examiner simply asserts that Babaoglu teaches "peer-to-peer application can be implemented in Java (page 7, section 4)" and that "it would have been obvious…to apply the teachings of Babaoglu to the system of Monday because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes." However, the Examiner has simply asserted "Babaoglu teaches 'peer-to-peer application' can be implemented in Java", and has not provided any argument or reference that Babaoglu teaches or suggests the actual claim limitations as recited in claim 9 of the instant application. The Examiner has not indicated where the cited references, specifically Babaoglu, teach member peers of a peer group in a peer-to-peer network environment.

Claims 14, 48 and 65

Claims 14, 48 and 65 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claims 1, 35, and 52.

Claims 15, 49 and 66

Claims 15, 49 and 66 depend from independent claims 1, 35, and 52, respectively, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 15, 49 and 66 for the following reasons.

The Examiner has failed to establish that the cited references, alone or in combination, teach wherein the system and the remote system are configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols for enabling the peer nodes to discover each other, communicate with each other, and cooperate with each other to form peer groups in the peer-to-peer environment.

The Examiner simply asserts that Babaoglu teaches "peer-to-peer application can be implemented in Java (page 7, section 4)" However, the Examiner has not provided any argument or reference that Babaoglu teaches or suggests the actual claim limitations as recited in claim 15 of the instant application. For example, the Examiner has not indicated where the cited references, specifically Babaoglu, teach systems configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols, nor has the Examiner indicated where the cited references teach a peer-to-peer platform protocol for enabling peer nodes to discover each other, nor has the Examiner indicated where the cited references teach a peer-to-peer platform protocol for enabling peer nodes to communicate with each other, nor has the Examiner indicated where the cited references teach a peer-to-peer platform protocol for enabling peer nodes to cooperate with each other to form peer groups in the peer-to-peer environment.

Claims 17, 51, and 68

Claims 17, 51 and 68 depend from independent claims 1, 35, and 52, respectively and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 1. In addition, Appellants traverse the rejection of claims 17, 51 and 68 for the following reasons.

The cited references fail to disclose, alone or in combination, program instructions executable by a processor to implement wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously.

The Examiner simply asserts that Babaoglu teaches "peer-to-peer application can be implemented in Java (page 7, section 4)" and that "it would have been obvious…to apply the teachings of Babaoglu to the system of Monday because it presents a framework supporting a new approach for building P2P application in which resource can be sharing by direct exchange between peer nodes." The Babaoglu reference describes:

> …Anthill, a framework to support the design, implementation and evaluation of P2P applications based on ideas such as multi-agent and evolutionary programming borrowed from CAS. An Anthill system consists of a dynamic network of peer nodes; societies of adaptive agents travel through this network, interacting with nodes and cooperating with other agents in order to solve complex problems. (Babaoglu, Abstract).

However, the Examiner has simply asserted "Babaoglu teaches 'peer-to-peer application' can be implemented in Java", and has not provided any argument or reference that Babaoglu teaches or suggests all of the claim limitations recited in claim 17 of the instant application. For example, the Examiner has provided no argument or reference, nor can the Appellants find anything, from the Babaoglu reference that teaches or suggests that the Babaoglu "Anthill" framework is a distributed computing system for

submitting computational tasks in a distributed heterogeneous networked environment or that the Babaoglu "Anthill" framework *decentralizes task dispatching and post-processing functions* and *enables a plurality of jobs to be managed and run simultaneously*.

Appellants note that the Examiner has not addressed the arguments regarding claims 17, 51, and 68.

Claims 18, 19, 20, 22, and 29-32

Appellants respectfully traverse the rejection of independent claim 18 for at least the reasons given above in regards to claim 1 and claim 17.

Appellants note that the Examiner has not addressed the arguments regarding claim 18.

Claim 21

Claims 21 depends from independent claim 18, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 18. In addition, Appellants traverse the rejection of claim 21 for the reasons given above in regards to claim 4.

Claim 23

Claims 23 depends from independent claim 18, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 18. In addition, Appellants traverse the rejection of claim 23 for the reasons given above in regards to claim 6.

Claim 24

Claims 24 depends from independent claim 18, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 18. In addition, Appellants traverse the rejection of claim 24 for the reasons given above in regards to claim 7.

Claims 25 and 27

Claim 25 depends from independent claim 18, and claim 27 depends from claim 25; therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 18. In addition, Appellants traverse the rejection of claim 25 for the reasons given above in regards to claim 8.

Claim 26

Claims 26 depends from claim 25, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 25. In addition, Appellants traverse the rejection of claim 25 for the reasons given above in regards to claim 9.

Claim 33

Claims 33 depends from independent claim 18, and therefore Appellants traverse this rejection for at least the reasons given above in regards to claim 18. In addition, Appellants traverse the rejection of claim 25 for the reasons given above in regards to claim 15.

CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-9, 11-27, 29-43, 45-60 and 62-68 was erroneous, and reversal of the decision is respectfully requested.

The Commissioner is authorized to charge the appeal brief fee and any other fees that may be due to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-65900/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Appellants

Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
(512) 853-8850

Date: December 16, 2008

VIII. CLAIMS APPENDIX

The following lists claims 1-9, 11-27, 29-43, 45-60 and 62-68, incorporating entered amendments, as on appeal.

1. A system, comprising:

a processor; and

a memory comprising program instructions, wherein the program instructions are executable by the processor to implement:

a virtual machine;

a default class loader for the virtual machine, configured to:

load classes for code executable within the virtual machine on the system from one or more local locations indicated by a class path of the default class loader;

determine that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path; and

generate an indication that the class is not loaded;

a remote class loader mechanism configured to:

detect the indication that the class is not loaded;

obtain the class from a remote system via a network; and

store the class in a location indicated by the class path of the default class loader on the system;

wherein the remote class loader mechanism is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader, and wherein the default class loader is independent from the remote class loader mechanism; and

wherein the default class loader is configured to load the class from the location indicated by the class path, and wherein the default class loader being configured to load the class from the location avoids class conflicts.

2. The system as recited in claim 1, wherein, to load the class from the location indicated by the class path, the default class loader is configured to:

locate the class stored in the location indicated by the class path; and

load the class from the location for access by the code.

3. The system as recited in claim 1, wherein the location is a default directory for storing remote classes.

4. The system as recited in claim 1, wherein the location is a user-specified directory for storing remote classes.

5. The system as recited in claim 1, wherein said indication is an exception generated by the code and indicating that the class is not stored in the one or more locations indicated by the class path.

6. The system as recited in claim 1, wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to send a message requesting the class to one or more remote systems, wherein the message comprises information about the class for identifying a class file on the remote system that comprises the requested class.

7. The system as recited in claim 1, wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to:

send a message requesting the class to the remote system; and

receive the class from the remote system in one or more messages in response to the message.

8. The system as recited in claim 1, wherein, to obtain the class from a remote system, the remote class loader mechanism is further configured to:

broadcast a message requesting the class to one or more remote systems including the remote system on the network; and

receive the class from the remote system in one or more messages in response to the broadcast message.

9. The system as recited in claim 8, wherein the one or more remote systems and the system are member peers of a peer group in a peer-to-peer network environment.

11. The system as recited in claim 1, wherein the virtual machine is a Java Virtual Machine (JVM).

12. The system as recited in claim 1, wherein the code is in a bytecode computer language.

13. The system as recited in claim 1, wherein the code is Java code.

14. The system as recited in claim 1, wherein the system and the remote system are peer nodes configured to participate in a peer-to-peer environment on the network.

15. The system as recited in claim 1, wherein the system and the remote system are configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols for enabling the peer nodes to discover each other, communicate with each other, and cooperate with each other to form peer groups in the peer-to-peer environment.

16. The system as recited in claim 1, wherein the code is a code fragment of an application configured for execution on the system, and wherein the remote system is a node in a distributed computing framework that comprises the application and is configured to provide computer-executable code fragments of the application to two or more other systems to run the code fragments in parallel to execute the application.

17. The system as recited in claim 1, wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously.

18. A distributed computing system, comprising:

a master node configured to provide computer-executable code fragments of an application to a plurality of worker nodes on a network, wherein the code fragments are configured to run tasks in parallel on two or more of the plurality of worker nodes to perform a job;

a worker node configured to receive a code fragment from the master peer node;

wherein the worker node comprises a virtual machine and a default class loader for the virtual machine, wherein the default class loader is configured to:

load classes for the code fragment executable within the virtual machine from one or more local locations indicated by a class path of the default class loader;

determine that a class needed to execute the code fragment is not stored in the one or more locations indicated by the class path;

wherein the worker node further comprises a remote class loader configured to:

detect that the class is not loaded;

obtain the class from a remote node via the network; and

store the class in a location indicated by the class path of the default class loader on the worker node;

wherein the remote class loader is configured to perform said detect, said obtain, and said store separate from and transparent to the default class loader, and wherein the default class loader is independent from the remote class loader; and

wherein the default class loader is further configured to load the class from the location indicated by the class path, and wherein the default class loader being configured to load the class from the location avoids class conflicts.

19. The distributed computing system as recited in claim 18, wherein, to load the class from the location indicated by the class path, the default class loader is configured to:

locate the class stored in the location indicated by the class path; and

load the class from the location for access by the code fragment.

20. The distributed computing system as recited in claim 18, wherein the location is a default directory for storing remote classes.

21. The distributed computing system as recited in claim 18, wherein the location is a user-specified directory for storing remote classes.

22. The distributed computing system as recited in claim 18, wherein, to detect that the class is not loaded, the remote class loader is further configured to detect an exception generated by the code fragment and indicating that the class is not on the worker node.

23. The distributed computing system as recited in claim 18, wherein, to obtain the class from a remote node, the remote class loader is further configured to send a message requesting the class to the remote node, wherein the message comprises information about the class for identifying a class file that comprises the requested class.

24. The distributed computing system as recited in claim 18, wherein the master node is the remote node, and wherein, to obtain the class from a remote node, the remote class loader is further configured to:

send a message requesting the class to the master node; and

receive the class from the master node in one or more messages in response to the message.

25. The distributed computing system as recited in claim 18, wherein, to obtain the class from a remote node, the remote class loader is further configured to:

broadcast a message requesting the class to one or more remote nodes on the network; and

receive the class from the remote node in one or more messages in response to the broadcast message.

26. The distributed computing system as recited in claim 25 wherein the one or more remote nodes, the worker node, and the master nodes are member peers of a peer group in a peer-to-peer network environment.

27. The distributed computing system as recited in claim 25, wherein the one or more remote nodes are worker nodes configured to receive code fragments from the master node.

29. The distributed computing system as recited in claim 18, wherein the virtual machine is a Java Virtual Machine (JVM).

30. The distributed computing system as recited in claim 18, wherein the code fragment is in a bytecode computer language.

31. The distributed computing system as recited in claim 18, wherein the code fragment is Java code.

32. The distributed computing system as recited in claim 18, wherein the worker node and the master node are peer nodes configured to participate in a peer-to-peer environment on the network.

33. The distributed computing system as recited in claim 18, wherein the worker node and the master node are configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols for enabling the peer nodes to discover each other, communicate with each other, and cooperate with each other to form peer groups in the peer-to-peer environment.

34. A system, comprising:

a default class loader means for a virtual machine, wherein the default class loader means is configured to load classes for code executable within the virtual machine on the system from one or more local locations indicated by a class path of the default class loader means;

means for determining that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path;

means for obtaining the class from a remote system via a network; and

means for storing the class in a location on the system indicated by the class path of the default class loader means;

wherein said means for determining, said means for obtaining, and said means for storing are configured to operate separate from and transparent to the default class loader, and wherein the default class loader means is

independent from said means for determining, said means for obtaining, and said means for storing; and

wherein the default class loader means is configured to load the class from the location indicated by the class path, and wherein the default class loader means being configured to load the class from the location avoids class conflicts.

35. A method, comprising:

loading classes for code executing within a virtual machine on a system from one or more local locations indicated by a class path of a default class loader for the virtual machine;

determining that a class needed to execute the code on the system is not stored in the one or more locations indicated by the class path;

generating an indication that the class is not loaded;

detecting the indication that the class is not loaded;

obtaining the class from a remote system via a network in response to said detecting;

storing the class in a location indicated by the class path of the default class loader on the system;

wherein said detecting, said obtaining, and said storing are performed separate from and transparent to the default class loader, and wherein the default class loader is independent from said detecting, said obtaining, and said storing; and

the default class loader loading the class from the location indicated by the class path, wherein the default class loading the class from the location avoids class conflicts.

36. The method as recited in claim 35, wherein said loading the class from the location indicated by the class path further comprises:

the default class loader locating the class stored in the location indicated by the class path; and

the default class loader loading the class from the location for access by the code.

37. The method as recited in claim 35, wherein the location is a default directory for storing remote classes.

38. The method as recited in claim 35, wherein the location is a user-specified directory for storing remote classes.

39. The method as recited in claim 35, said indication is an exception generated by the code and indicating that the class is not stored in the one or more locations indicated by the class path.

40. The method as recited in claim 35, wherein obtaining the class from a remote system comprises sending a message requesting the class to one or more remote systems, wherein the message comprises information about the class for identifying a class file on the remote system that comprises the requested class.

41. The method as recited in claim 35, wherein said obtaining the class from a remote system comprises:

sending a message requesting the class to the remote system; and

receiving the class from the remote system in one or more messages in response to the message.

42. The method as recited in claim 35, wherein said obtaining the class from a remote system comprises:

broadcasting a message requesting the class to one or more remote systems on the network; and

receiving the class from the remote system in one or more messages in response to the broadcast message.

43. The method as recited in claim 42, wherein the one or more remote systems and the system are member peers of a peer group in a peer-to-peer network environment.

45. The method as recited in claim 35, wherein the virtual machine is a Java Virtual Machine (JVM).

46. The method as recited in claim 35, wherein the code is in a bytecode computer language.

47. The method as recited in claim 35, wherein the code is Java code.

48. The method as recited in claim 35, wherein the system and the remote system are peer nodes configured to participate in a peer-to-peer environment on the network.

49. The method as recited in claim 35, wherein the system and the remote system are configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols for enabling the peer nodes to discover each other, communicate with each other, and cooperate with each other to form peer groups in the peer-to-peer environment.

50. The method as recited in claim 35, wherein the code is a code fragment of an application configured for execution on the system, and wherein the remote system is a node in a distributed computing framework that comprises the application and is configured to provide computer-executable code fragments of the application to two or more other systems to run the code fragments in parallel to execute the application.

51. The method as recited in claim 35, wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously.

52. A computer-accessible storage medium, comprising program instructions, wherein the program instructions are computer-executable to implement:

loading classes for code executing within a virtual machine on a system from one or more local locations indicated by a class path of a default class loader for the virtual machine;

determining that a class ~~is~~ needed to execute the code on the system is not stored in the one or more locations indicated by the class path;

generating an indication that the class is not loaded;

detecting the indication that the class is not loaded;

obtaining the class from a remote system via a network;

storing the class in a location indicated by the class path of the default class loader on the system;

wherein said detecting, said obtaining, and said storing are performed separate from and transparent to the default class loader, and wherein the default class loader is independent from said detecting, said obtaining, and said storing; and

the default class loader loading the class from the location indicated by the class path, and wherein the default class loader loading the class from the location avoids class conflicts.

53. The computer-accessible storage medium as recited in claim 52, wherein, in said loading the class from the location indicated by the class path, the program instructions are further computer-executable to implement:

the default class loader locating the class stored in the location indicated by the class path; and

the default class loader loading the class from the location for access by the code.

54. The computer-accessible storage medium as recited in claim 52, wherein the location is a default directory for storing remote classes.

55. The computer-accessible storage medium as recited in claim 52, wherein the location is a user-specified directory for storing remote classes.

56. The computer-accessible storage medium as recited in claim 52, wherein said indication is an exception generated by the code and indicating that the class is not stored in the one or more locations indicated by the class path..

57. The computer-accessible storage medium as recited in claim 52, wherein, in said obtaining the class from a remote system, the program instructions are further computer-executable to implement sending a message requesting the class to one or more remote systems, wherein the message comprises information about the class for identifying a class file on the remote system that comprises the requested class.

58. The computer-accessible storage medium as recited in claim 52, wherein, in said obtaining the class from a remote system, the program instructions are further computer-executable to implement:

sending a message requesting the class to the remote system; and

receiving the class from the remote system in one or more messages in response to the message.

59. The computer-accessible storage medium as recited in claim 52, wherein, in said obtaining the class from a remote system, the program instructions are further computer-executable to implement:

broadcasting a message requesting the class to one or more remote systems on the network; and

receiving the class from the remote system in one or more messages in response to the broadcast message.

60. The computer-accessible storage medium as recited in claim 59, wherein the one or more remote systems and the system are member peers of a peer group in a peer-to-peer network environment.

62. The computer-accessible storage medium as recited in claim 52, wherein the virtual machine is a Java Virtual Machine (JVM).

63. The computer-accessible storage medium as recited in claim 52, wherein the code is in a bytecode computer language.

64. The computer-accessible storage medium as recited in claim 52, wherein the code is Java code.

65. The computer-accessible storage medium as recited in claim 52, wherein the system and the remote system are peer nodes configured to participate in a peer-to-peer environment on the network.

66. The computer-accessible storage medium as recited in claim 52, wherein the system and the remote system are configured to participate as peer nodes in a peer-to-peer environment on the network in accordance with one or more peer-to-peer platform protocols for enabling the peer nodes to discover each other, communicate with each other, and cooperate with each other to form peer groups in the peer-to-peer environment.

67. The computer-accessible storage medium as recited in claim 52, wherein the code is a code fragment of an application configured for execution on the system, and wherein the remote system is a node in a distributed computing framework that comprises the application and is configured to provide computer-executable code fragments of the application to two or more other systems to run the code fragments in parallel to execute the application.

68. The computer-accessible storage medium as recited in claim 52, wherein the system and the remote system are configured to participate in a distributed computing system on the network for submitting computational tasks in a distributed heterogeneous networked environment that utilizes peer groups to decentralize task dispatching and post-processing functions and enables a plurality of jobs to be managed and run simultaneously.

IX. EVIDENCE APPENDIX

No evidence submitted under 37 CFR §§ 1.130, 1.131 or 1.132 or otherwise entered by the Examiner is relied upon in this appeal.

X. RELATED PROCEEDINGS APPENDIX

There are no decisions on any related proceedings.